**概述**

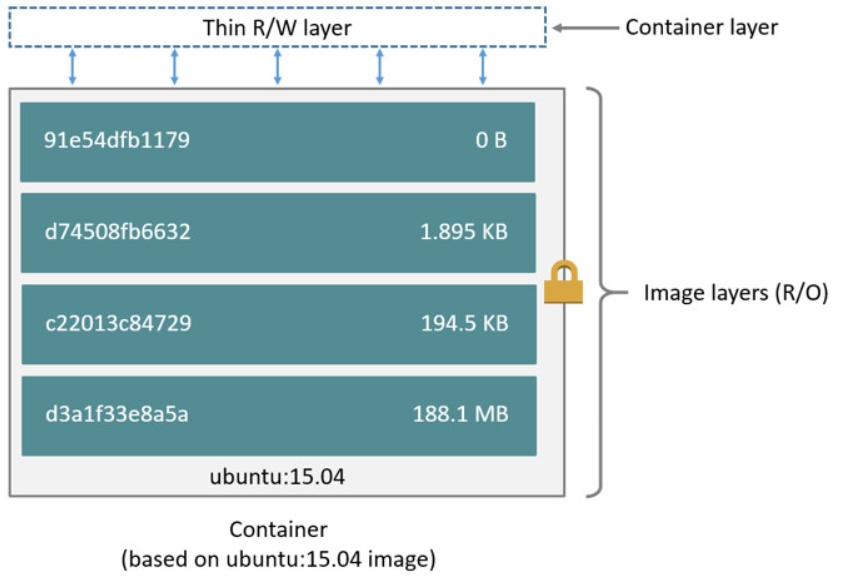
Docker镜像方便了Docker的部署，开箱即用，是Docker之所以流行的关键所在。但是其动辄几百兆的镜像大小也让人非常头大，尤其现在的Paas平台，少则几十，多则上百的镜像文件让Paas平台的输出和部署非常头大，动辄几十G的数据量，镜像占了绝大多数，占用大量磁盘资源不说，如果远程部署带宽不够的话，光数据的传输传个几天几夜都是稀松平常的事。

很多Docker使用者都不注意镜像大小问题，每个人都觉着自己应用的Docker镜像几十兆、几百兆，并不会占用多少资源。但当很多个应用集成在一起，对外整体发布输出的时候就出现了数据大到没法网络传输的问题。本文旨在说明镜像原理，镜像制作过程中如何避免垃圾数据占据镜像的“空间”，如何压缩镜像等。

**Docker镜像原理**

Docker镜像是分层的，通过Union Mount技术将各层“捏合”在一起形成镜像。目前，Docker镜像默认使用的Union Mount技术是Overlayfs，一种类似aufs的一种堆叠文件系统，于2014年正式合入Linux-3.18主线内核。

在Docker中，为了节省空间，同时也为了Docker容器能多次重复创建，将Docker的存储分成了镜像层和容器层，容器层设置为读写模式，而镜像层是只读的，从下往上，以栈的方式组合在一起。镜像层既做镜像又供容器运行时使用，当容器启动时，Docker会创建一个可读写的容器层，把这个容器层添加在镜像层上。



1-1 容器和镜像结构示例8

Docker在制作镜像时也是使用的分层，会将新的镜像层按照“写时复制（copy-on-write）”策略写入新的数据，然后使用Union Mount技术将新层和基础镜像整合形成新的“只读（read-only）”镜像。由于使用了这种分层然后Union Mount的方式，所以，当一个镜像创建完成后，那么以此为基础镜像的“子”镜像只能越来越大，这也是影响镜像大小一个很重要的原因。

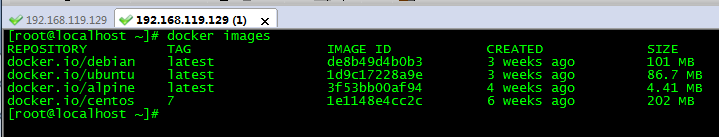
当然，分层也有分层的好处，就是不同的镜像，如果存在相同的层，那么可以只存储一份数据，共享一个层，然后通过“写时复制”的方式将动态数据提取到容器当中，只有在容器运行时候数据才会被复制一份到容器层做修改，所有操作写操作都发生在容器层当中，这样既节约了空间，又大大提高了容器的启动速度。

**如何减小镜像“体积”**

1.选择较小的基础镜像

在制作镜像的时候，如果旧的镜像过大，那么以此为基础制作的镜像也只能只大不小，我们可以选用合适的更小的基础镜像，然后按照自己的需求进行添加层。

下图中，我们可以看到不同的基础镜像大小，可以想象，如果在alpine镜像和centos镜像都满足要求的情况下，结果我们选择了centos镜像，这对镜像体积的影响将会是多么大。



1-2 镜像大小比较

2.减少镜像层数

在镜像制作书写Dockerfile的时候，如果太多的使用RUN指令，经常会导致镜像有特别多的层，镜像很臃肿，而且甚至会碰到超出最大层数（127层）限制的问题，因此，我们应该把多个命令串联合并为一个 RUN，每一个 RUN 要精心设计，确保安装构建最后进行清理，这样才可以降低镜像体积，以及最大化的利用构建缓存。

3.避免垃圾数据

制作镜像时应尽量避免使用apt-get或yum，即使使用了也应该在安装结束后清理掉缓存，因为apt-get和yum在执行安装的时候不加特殊参数会将安装包缓存到本地，增加镜像体积。同时注意，在清理的时候，应该将清理命令和安装命令放在一起执行，即放在同一层内执行，否则，根据写时复制的策略，一层清理另一层的数据是无效的。

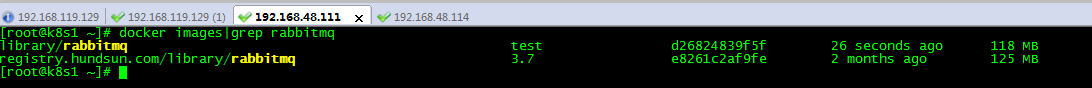
如果我们使用了其他方式，比如源码包安装或者已经已经编译好的程序直接复制挂载到镜像层形成新镜像的时候，我们应该尽量删除冗余信息，比如一些源码文件、log日志、说明文档等占用磁盘空间却对程序运行无用的文件。

4.压缩镜像

我们可以使用Docker自带命令，export和import来协助压缩镜像，使用这种方式需要先将容器运行起来，但这个过程中会丢失镜像原有的一些信息，比如：导出端口，环境变量，默认指令。所以，这个压缩方式要慎用，一般不推荐。

docker **export** 9faf3c391fcb | docker **import** - **library**/rabbitmq:test

9faf3c391fcb是container的ID，压缩效果如图：

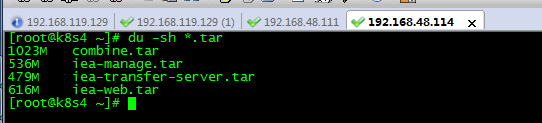


1-3 压缩结果比较

Docker还提供了一种镜像压缩方式，就是在build镜像的时候使用--squash参数，不过该功能自1.13版本加上，还处于测试阶段，生产环境慎用。

5.统一基础镜像

在互联网企业中，大多为java应用，如果不同的部门采用完全不同的镜像，各自为政，当大家应用集成在一起输出的时候由于镜像的各个层完全不同导致镜像库会非常臃肿庞大。公司可以为各个部门制作共同使用的基础镜像，然后制定统一的镜像和应用打包发布规范，这样既减少了docker镜像仓库的“体积”，也大大增加了数据在传输拷贝时候的效率。



1-4 镜像导出

如图中，三个镜像大小分别为536M、479M、616M，合计1631M，三个镜像只共用了一个JDK镜像，将三个镜像合在一起输出时候为1023M，减少了608M的大小。而且，这三个镜像是有可以继续压缩的空间的，三个应用皆为Jboss应用，但Jboss却是三个镜像各自作为层挂载到自己的镜像当中的。如果我们制作一个通用的Jboss镜像，然后再把应用打包进去，那么三个镜像除了本身应用的内容，其他将全部共用，镜像总大小将会继续压缩几百兆。

**后记**

Docker技术是一个让人感觉非常惊奇的设计理念，其快速简洁可重复的应用部署方式大大减少了运维工作量。但在使用过程中发现，在镜像问题上花费的时间是最多的，尤其是由于镜像太大导致无法通过网络拷贝或者是拷贝时间长到让人无法忍受，于是，总结了此文以记之。